CuEira, gene-enviroment interaction analysis on GPU

Daniel Berglund

April 2014

Abstract

Abstract in english

Abstract

Abstract pÃě svenska

Acknowledgements

Asdf

Contents

1 Introduction **1**
1.1 Genome-wide association studies 1
1.2 GEIRA and JEIRA 1
1.3 Defining Interaction 2

2 Background **3**
2.1 Mathematical Background 3
2.1.1 Confounders and Covariates 4
2.1.2 The Multiple Testing Problem 4
2.1.3 Contingency Tables 5
2.1.4 Logistic Regression 5
2.1.5 Data Mining and Machine Learning Approaches 7
2.2 Computer Architecture 9
2.2.1 CPU 9
2.2.2 Caching and Memory 9
2.2.3 Concurrency and Threads 10
2.2.4 Accelerators, GPU and Xeon Phi 10
2.2.5 Which program for GPU and which for CPU? 10
2.2.6 Clusters 12
2.2.7 CUDA programming model 12
2.2.8 Performance Measures 18
2.3 Software Design 22
2.3.1 Unit Tests and Mocks 22
2.3.2 Design Patterns 23

3 Algorithm **24**
3.1 Current algorithm 24
3.2 CuEira 24

4 Results **25**

5 Discussion and Conclusions **26**

6 Outlook **27**

7 Appendix **28**
7.1 A List of Variables 28
7.2 B List of Abbrivations 28

Bibliography **29**

Chapter 1

Introduction

1.1 Genome-wide association studies

Within Epidemiology genome-wide association studies(GWAS) are a type of study to search for associations between genetic markers and diseases. Classically GWAS does not consider interactions between the genetic markers themselves or with environmental factors. Investigating gene-gene interactions recently has become more common[1], however gene-environment interactions are still an uncommon focus in studies[2]. Interactions between genes and environmental factors are considered to be important for complex diseases such as cancer and autoimmune diseases.[1, 2, 3, 4]

GWAS are usually either cohort or case-control. In cohort studies a healthy sample of a population is being followed and over time some individuals will developed the disease of interest. In case-control studies two groups are compared to find risk factors. One group consists of individuals with the disease and the other of individuals that are similar to the cases but who do not have the disease.[5, 6]

The genetic markers are commonly single-nucleotide polymorphisms(SNPs). SNPs are variations in the genome where a single nucleotide is different between individuals in a population[7]. Environmental factors can be various things such as smoking, physical activity and so on. The amount of data to handle usually consists of thousands individuals and hundred thousands or millions of SNPs. Due to the high number of SNPs few algorithms are capable to investigate more than second order interaction in a reasonable time. There are some algorithms that can handle higher interaction orders however these have drawbacks[8, 9, 10].

1.2 GEIRA and JEIRA

GEIRA is a tool for analysing gene-gene and gene-environment interaction. It uses logistic regression and causal interaction[3]. *JEIRA* is a parallel implementation of the environmental interaction analysis in GEIRA.

1.3 Defining Interaction

There are several ways to define interaction. The overall goal is usually to find if *biological* interactions are present. *Biological* interaction is when the factors cooperate through a physiological or biological mechanism and causes the effect, e.g. disease. This information can be used to explain the mechanisms involved in causing the disease and possibly help to find cures for them. However biological interaction is not well defined and thus it is not possible to calculate it directly from data.[5, 11]

Statistical interaction on the other hand is well defined. However it is scale dependent, ie. interactions can appear and disappear based on transformations of the data. Statistical interaction also depends on the used model. The common way to define statistical interaction is to consider the presence of a product term between the variables in the statistical model, this is referred to as *multiplicative* interaction. For instance for a linear model

$$f(x, y) = ax + by + cxy + d \tag{1.1}$$

c is the product term that represents multiplicative interaction between variables x and y. Statistical interaction is often referred to as just interaction which can make it a bit confusing.[3, 5]

It can also be defined as the divergence from additive effects on a logarithmic scale, e.g.

$$OR_{both\,factors\,present} > OR_{first\,factor\,present} + OR_{second\,factor\,present} - 1 \tag{1.2}$$

It is called additive interaction. It implies biological interaction as defined by Rothman, which is sometimes called *causal interdependence* or *causal interaction.*[5]

Chapter 2

Background

Most of the conducted research on interactions in GWAS has been focused on gene-gene interactions. Few studies and tools have been investigating gene-environment interactions.
Both CPU clusters[12] and GPUs[8, 13, 14, 15, 16, 17] have been used for GWAS. GPUs have been a popular choice since most of the methods for searching for interaction are good on GPUs because each combination can usually be considered independently from the others. More about this in section 2.2.5

2.1 Mathematical Background

There are a lot of algorithms and programs proposed for searching for interaction, most have focused on gene-gene interaction as already mentioned. They can be roughly classified into four categories, exhaustive, stochastic, machine learning/data mining and stepwise[10]

SNPs are commonly binary variables with either dominant or recessive models. Dominant is when a person only has to have one allele affected to get the trait. Recessive is when both alleles have to be the cause. The case where the risk is present is coded as 1 and absence is coded as 0. Co-dominant model can also be used and in that case when both alleles have the risk factor is coded as 2, when only one is present is coded as 1 and absence is 0 as before. That SNPs can be viewed as binary is an advantage that is almost always used when searching for gene-gene interactions.

Environmental factors can be of any type[2]. Gene-gene interaction tools can sometimes be used to find gene-environment interaction, however they usually require the variable to be binary or have other problems since they weren't designed for environmental interaction.[2]

Exhaustive search is the most direct approach, it compares all combinations of the SNPs in the data. Exhaustive search methods will not miss a significant combination because it didn't consider it however it also means that they can be slow. Multifactor-Dimensionality Reduction(MDR)[18] and BOOST[19] are two examples of this type of algorithm.

Stochastic methods uses random sampling to “walk” through the data. BEAM[20] is one example and it uses Markov Chain Monte Carlo(MCMC) method.

Data Mining and *Machine Learning* are methods that learn from data and tries to generalize it. MDR[18] is a type of data mining method and is among the most common methods used in GWAS. See section 2.1.5 for more details.

Stepwise methods uses a filtering stage and a search stage. At the filtering stage uninteresting combinations are filtered out by using some exhaustive method. The other SNPs are the examined more carefully in the search stage. BOOST[19] is an example which uses clever data structures and a likelihood ratio test to filter the data before applying log-linear models.

2.1.1 Confounders and Covariates

Confounding is one of the central issues in the design of epidemiological studies. It is when the effect of the exposure is mixed with the effect of another variable. So if we do not measure the second variable the first effect would be estimated as stronger than it really is. The second variable is then a *confounder*. Several methods in epidemiology are about avoiding or adjusting for confounding. Sometimes these variables needs to be incorporated into the models. *Covariates* are possible confounders or other variables that one wants to adjust for in the model. Sometimes covariates are called *control variables*.[11, 5]

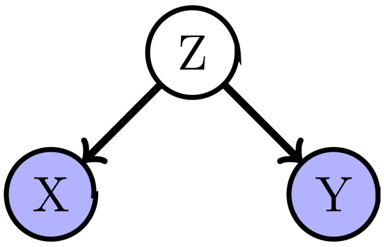


Figure 2.1: *Illustration of a simple case of confounding. If we do not observe Z we might falesly find an association between X and Y. Wikipedia Commons*

2.1.2 The Multiple Testing Problem

It is not uncommon to test a lot of hypothesis on the same data and in the case of GWAS it can be billions of tests. If one continues to test one should eventually find something that is significant. Also it is important to remember that the standard p-value of 5% means that we expect to find 1 in 20 false positives under the assumption that the null hypothesis is true. The problem arises from the fact that the hypothesis tests are dependent on each other since they use parts of the same data. This is the multiple testing problem and if it is not corrected for one might find a lot of false positives.[21]

Bonferroni correction is the simplest and viewed as the most conservative way to correct for this problem. It simply divides the p-value threshold that a test passes by the number of hypothesis. However the number of hypothesis made is not always clear. With a two-stage analysis, is the number of hypothesis the number of tests done in both stages combined, the number made in the first stage or the second stage?[21]

2.1.3 Contingency Tables

A contingency table is a matrix used to describe categorical data. Each cell contains the frequency of occurrences with a specific combination of variables. Table 2.1 is an example of an 2 x 3 table. From this table we can for instance see that 171 persons that got the placebo had a non-fatal attack. Contingency tables are the basis for various statistical tests to model the data. Contingency tables can be used directly for tests like chi^2.[22]

	Fatal Attack	Non-fatal Attack	No Attack
Placebo	18	171	10 845
Aspirin	5	99	10 933

Table 2.1: *Contingency table describing the outcome of a medical study, from [22]*

2.1.4 Logistic Regression

One way to model the contingency tables is by using logistic regression. Logistic regression is a type of linear regression model for classification that models a latent probability for the outcomes. The outcomes are binary, however the method can be extended to multiple outcomes. In this work we will only consider them as binary. Logistic regression transforms the probability by using the *logit* transformation. The logit transformation with probability p is

$$log(\frac{p}{1-p}) \tag{2.1}$$

The probability with a set of predictor variables X is $\pi(X) = P(Y = 1)$. The linear regression model with n predictors $X = (x_1, x_2, ..., x_n)$ and by using the logit transformation is then

$$logit[\pi(X)] = \alpha + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n \tag{2.2}$$

If move the logit to the right side we get the modelling of the probability

$$\pi(X) = \frac{exp(\alpha + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n)}{1 + exp(\alpha + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n} \tag{2.3}$$

Finding the coefficients are done in a similar way as other linear regression models since they all are generalized linear models. It's done using maximum likelihood(ML), often via Newtons method.
It's an iterative method which means it is slow compared to non iterative methods. Logistic regression as most other methods for interaction assumes independence between the variables.

2.1.4.1 Statistic Measures

There are various statistical measures of interest that we can calculate from the logistic regression model.

Odds ratio(OR) describes the change in odds between different levels of exposure.
The odds is ?
In binary case it simplifies to the increased odds when exposed.
An odds ratio of one means the the odds and therefore the odds is the same whether exposed or not. An odds ratio below one means that the odds of being affected is lower when exposed and if the odds ratio is above one it means that the odds is increased. In the case when the outcome is an disease or similar variables with odds ratio below one are called *protective* and when it is above one it is a *risk factor*.
Relative risk(RR)

$$RR = \frac{p_{event\ when\ exposed}}{p_{event\ when\ not\ exposed}} \tag{2.4}$$

RR can not be estimated in case control since it requires the prevalence in whole population which you need cohort studies to find. In case control it is approximate with OR.
Relative excess risk due to interaction(RERI) is

$$RERI = RR_{11} - RR_{10} - RR_{01} + 1 \tag{2.5}$$

Attributable proportion due to interaction(AP)

$$AP = \frac{RERI}{RR_{11}} = \frac{1}{RR_{11}} - \frac{RR_{10}}{RR_{11}} - \frac{RR_{01}}{RR_{11}} + 1 \tag{2.6}$$

Synergy index(SI)

$$\frac{RR_{11} - 1}{RR_{10} + RR_{01} - 2} \tag{2.7}$$

SI is simpler to investigate on the log scale

$$ln(SI) = ln(RR_{11} - 1) - ln(RR_{10} + RR_{01} - 2) \tag{2.8}$$

Lack of interaction shows in the values by AP=0 SI=1

2.1.4.2 Minor Allele Frequency and Risk Allele

Minor allele frequency(MAF) is the frequency of the least common allele. If it is too low an analysis will be of little value, 5% is usually used as the threshold. The same problem occurs if the absolute number is too low.
The *(*risk allele) is the allele that is most frequent in cases but also more frequent in cases than controls

2.1.4.3 Coding the interaction variable

To represent the interaction in the model there needs to be a variable that holds the information. This value depends on the chosen model for interaction as discussed in ?.

However there is a problem when there is multiple categories in both variables. In short it's becase of how OR works and that the categories for LR become rated in increased or decreased order, that is how the categories are labeled matters. Say three locations, code as 1,2,3 or 2,1,3 matters.
There is two cases, one were we don't know any severness or whatever, say locations
second were we do have it, say it's smoking as never, previous and current. current is likely worse than previous worse than never
but if the second one also has levels say it's also of second type as simplest case example here how do we know which is worst? current smoker or exampel badest number of possiblitys increases with levels and to examine all means lots of computions and loss of statistical powers
Why occurs due to how OR works
However in our case we know that the gene is binary under dominat or recessive model. So we can just multiply. However means we can't use co-domiant model. There are ways around it by using dummy variables or stratification but that will decease statistical power.

2.1.4.4 Recoding

2.1.5 Data Mining and Machine Learning Approaches

Approaches based on Data Mining and Machine Learning have been a popular choice for GWAS. Multifactor-Dimensionality Reduction(MDR)[18] and Random Forest(RF)[23] are among the most common ones[2, 1]. There are other methods as well such as clustering approaches [10]. Most of them are used for screening the data for possible interactions[2, 1].

Their biggest advantage is that they are usually nonparametric and designed with high dimensional data in mind. However they are prone to overfitting and the usual way to try to prevent that is to use cross validation and sometimes permutation tests. It means that even if the method itself is fast it is repeated so many times that the whole algorithm can be slow in the end.[1]

2.1.5.1 Multifactor-Dimensionality Reduction

Multifactor-Dimensionality Reduction(MDR) is a method that reduces the number of dimensions by combining several dimensions into one. For GWAS it combines a number of variables from all the variable combinations and its new dimension is then compared against the outcome. If its predictability is high then the variables that were combined are considered to interact with each other. MDR repeats the process on all combinations of variables. It combines the selected n variables by calculating the ratio of cases versus controls for each combination of the possible values of the variables. If the ratio is above a certain threshold all the members of that groups is gets the value 1 for the new dimension, otherwise 0. However due to the cross validation and permutation tests this method can be slow. On the other hand it is still faster than exhaustive search with regression methods.[1, 18]

An simple example of MDR using exclusive or(XOR). XOR is an logical operator that is true if one and only one of its two variables is true. We have 4 possible combinations and an occurrence for each of them. The combination (1,0) and (0,1) both have one case with outcome 1 so MDR will classify them as 1 in the new variable Z. The other two combinations have outcome 0 so will be classified with Z=0. From here it is easy to make an predictor from Z to the outcome Y by comparing the values.

Y	$\mathbf{X_1}$	$\mathbf{X_2}$
1	1	0
1	0	1
0	0	0
0	1	1

Table 2.2: *XOR table with outcome* **Y** *and variables* $\mathbf{X_1}$ *and* $\mathbf{X_2}$.

Y	**Z**
1	1
1	1
0	0
0	0

Table 2.3: *XOR table with* $\mathbf{X_1}$ *and* $\mathbf{X_2}$ *combined into* **Z** *using MDR.*

MDR can been used for gene-environment interaction but requires modifications since MDR can only handle binary variables. There are extensions that can use continues variables, however these are regression based so these will be slower than regular MDR.[2]

2.1.5.2 Random Forest

Random Forest(RF) is an ensemble learning method. Ensemble methods combine multiple models to improve performance. RF takes randomized samples of the data and builds decision trees on each of them. These trees are then combined to form the classifier. Usually hundreds or thousands of trees are used depending on the problem[23]. One of the most popular variants of Random Forest for GWAS is Random Jungle[24].

It has been shown in high dimensional data that RF tends to only rank interacting factors high if they have strong marginal effects[25]. Also the ranking of the variables does not indicate which factor it is interacting with either since it is based on the joint distributions[2]. How to incorporate the environmental factors in RF is also not obvious and using variables with very different scales can bias RFs results[2].
Decision tree figure here

2.2 Computer Architecture

Most computers today have a multi-core architecture, e.g. their CPUs have multiple cores.
means several processors share the main memory and other resources. they can cooperate on the same problem but it they are coordinated on a case by case basis since each algorithm requires different sharing.
Von Neumann architecture It was first used in EDVAC which was one of the first stored program computers[26]. Stored program computers uses electronic memory to store the program instructions[27].

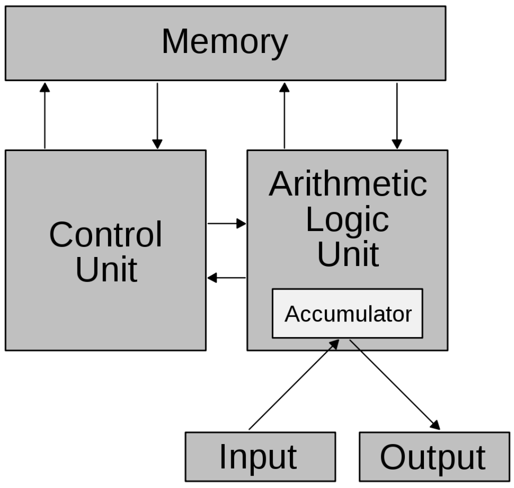


Figure 2.2: *Schematic of the Von Neumann architecture. Wikipedia Commons*

some text here about von neaumann architecture
For HPC it is easiest to think about the architecture as three main parts, data storage, processor and the memory. More text here[28]

2.2.1 CPU

Central processing unit(CPU) is [29]
Until 1996 there was computers with just a single core CPU on top 500[30]. Since about 2005 most desktop computers CPU has several cores which has increased the need for parallelization. The cores themselves aren't getting faster, the CPUs have more cores.
As you can see most of it is not used for calculations. A large part of the area is used for optimizations and caching. We will talk about caching in the next section. CPUs have various optimizations too improve their performance.
branch prediction

2.2.2 Caching and Memory

The CPU optimzation that is most important to know for programmers is caching because it affect the way the code has to be written and the diffrence between using it correctly and not is huge.

It happens when two pointers point to the same place in memory and the code does operations on them at the same place. This means that operations that look like they can be done in any order actually can't and thus prevents the compiler to make optimizations. The problem is that the compiler has to take this in to account for all pointers even if no aliasing occurs because it might happen.
[31]

2.2.3 Concurrency and Threads

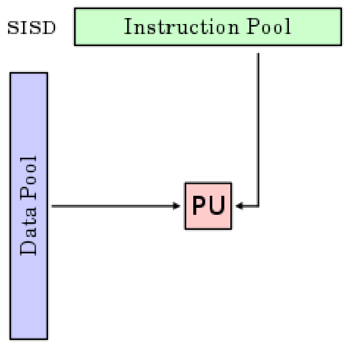
concurrency what is it
doing multiple things and the same time, instead of *sequential*
dinning philosphers
in computer science, instructions seperated in threads
The architectures can be divided into groups. The non parallelized is *SISD*(Single-Instruction,Single-Data) and is what the name says, single instructions on single data. *SIMD*(Single-Instruction, Multiple-Data) is applying the same instruction on different data. SIMD is also called vectorisation. *MIMD*(Multiple-Instruction, Multiple-Data) applies different instructions on different data.
There is also *MISD*(Multiple-Instruction, Single-Data) but it is uncommon. It performs different operations at the same time on a piece of data.
Deadlocks occur when two or more threads wait for the other to finish, since they are all stucking in waiting the program will not progress.
A *race condition* occurs when the result depends on which threads affects it first. Due to the nature of how parallelism works the output can vary between different runs of the program depending on which thread is fastest.
Locks are used to prevent situtions as the ones described above by making sure only one thread at a time can change the data in question.
[29]

2.2.4 Accelerators, GPU and Xeon Phi

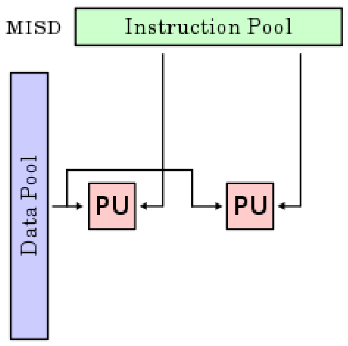
Graphics processing unit(GPU) have become more popular for general computing the last 10 years.
'dummer' threads than CPU, more area for calculations
Xeon Phi, intel mic mimd knights landing, interesting new development from intel. however usually but not always require memory copy still need vectorisation and other stuff you end up with similar problems as gpu, though probably easier to use
structure of mic
Most CPUs have a marginal decrease in performance when using double precision instead of single. GPUs lose much more.

2.2.5 Which program for GPU and which for CPU?

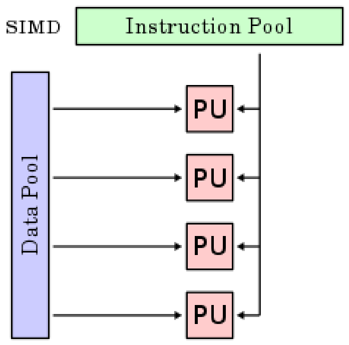
As with a lot of things it depends on the algorithm to implement etc.
If you have lots of legacy code then then you might need to rewrite a lot. However by using a profiler the parts that take up the most time can be found and moved to the GPU. However that migh be hard if the program isn't properly structured. No legacy code can also be a good thing. Old code might not be



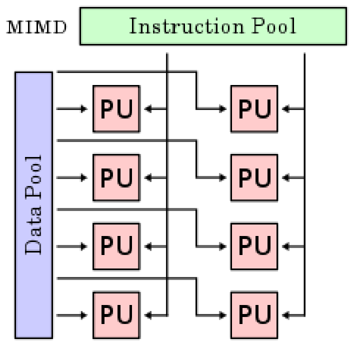
(a) *SISD*



(b) *MISD*



(c) *SIMD*



(d) *MIMD*

Figure 2.3: *Flynns taxamony*

written with modern standards or use new nifty features and therefore lose speed.
Xeon Phi is an alternative, simpler but can still be tricky and require memory copy.
GPUs loses more when going from single precision to double than CPUs. Peak performance on tesla-kepler cards for single precision is around 4 Tflops while double precision is around 1.4 Tflops. [33]
ref the table, single is much faster than double
In short GPUs excel at linear algebra but Intel MIC is probably a good alternative for it to.
GPUs are good for interaction algorithms since most approaches are embarrassingly parallel since they consider each combination independently from the others. They also often perform a lot of linear algebra. Several studies got high gains from implementing their programs on GPU, however their CPU versions weren't parallelized so it is likely that the gains would have been less compared to an optimized and parallelized CPU version.[8, 13, 14, 15, 16, 17] However one study made a CPU cluster version and a GPU version of an algorithm that uses chi^2 tests, they found that 16 CPU nodes had the same performance as a single GTX 280 card[34].

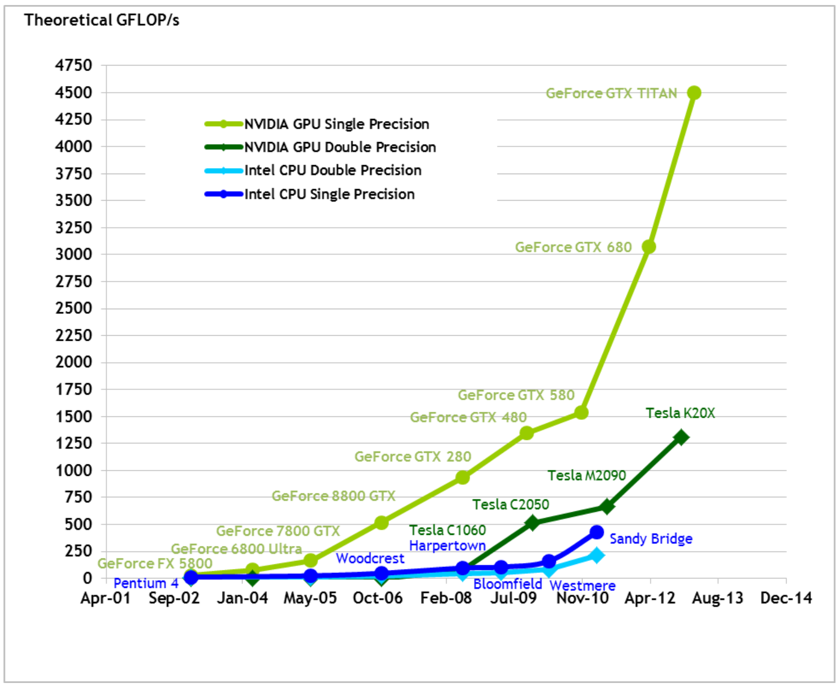


Figure 2.4: *Bla bla [32]*

2.2.6 Clusters

Clusters are collections of computers that can work together in several ways and are so tightly connected that they can usually be viewed as a single system. They communciate over a shared network but have separate memory and processors. Storage is usually shared. A computer in a cluster is called a *node*. [28, 35]
various programming models, shared memory and so on
mpi is commonly used for clusters[35]
top 500[30]
Few clusters used GPUs before 2009 but now mix of GPUs and cpu are common among the top clusters. It was driven largely by demand for power saving while still giving high performance. The GPUs can do the heavy computation while other parts are used on CPU. Most of these clusters are highly ranked in Green 500. [35]
Flops is not everything, bad performance compare to theoretical maximum Stuck in network, hard drives, etc. Latency Very few programs scale to the fastest clusters, peta scale.
Tianhe 2 is current top. Theoretical performance 54.9 PetaFlops.[30] Max achived with LinPack 33 PetaFlops 60% efficiency
Picture of Tianhe-2 here

2.2.7 CUDA programming model

Cuda function calls are done by using kernels divided into blocks
These blocks can be executed in any order so depending on the number of streaming multiprocessors(SM) they get executed differently

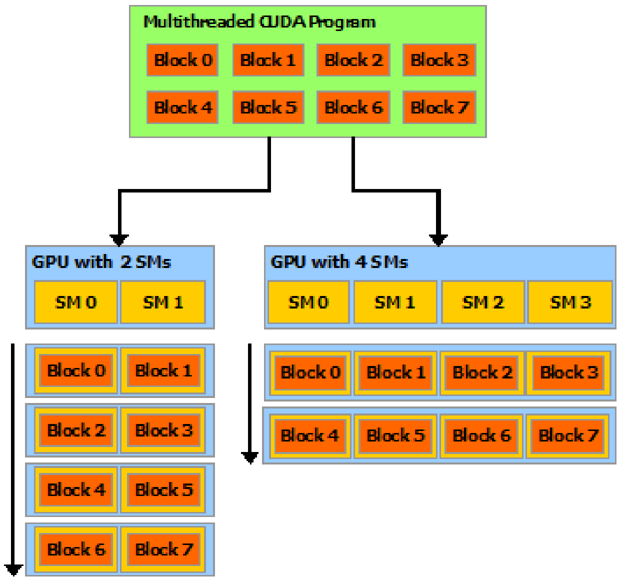


Figure 2.5: *Text abou blocks. Need to make a black white version*

Kernels are declared by using the ___global___ specifier setting stuff for the call to the kernel by using «<...»>
The blocks are organized in one, two or three dimensional *grid*. These can be accesed in a kernel. each block also share some memory, see efficient cuda part or the cuda programming guide for the details.
The streaming multiprocessors can execute hundreds of threads concurrently, it uses *SIMT*(Single-Instruction, Multiple-Thread). It is similar to SIMD but there are some differences. It executes on groups of 32 threads called warps when an SM gets a number of blocks they are split into warps that are assigned to warp schedulers each warp scheduler gives one instruction to the whole warp so full efficiency is achieved when all 32 threads do the same instruction. if there is any divergence it has to disable unrelated threads, so divergence is costly. different warps can diverge without problem
since the GPUs memory is physically on a different device, the *host* and *device* have separate memory spaces. They are referred to as *host memory* and *device memory*. Because it is separate data has to be transferred to the device and this is done by explicit calls to transfer sections of the memory.
The kernel calls can be made on a stream. By default a kernel is executed on the null stream. The easiest way to think of the stream is as queue, the kernels on a stream will be executed in the order they are made but different kernels from different streams can be executed in any order and if there are resources enough at the same time. In this way it is possible to execute up to 32 concurrent kernels depending on GPU.
Streams are important for asynchronous transfers. These transfers are executed on a stream and just as kernels gets executed after the previous kernels on the same stream is done. And as with the kernels the transfer on one stream can happen . This however depends on the number of copy engies and kernel engines. Some older GPUs have one copy engine and one kernel engine. Newer

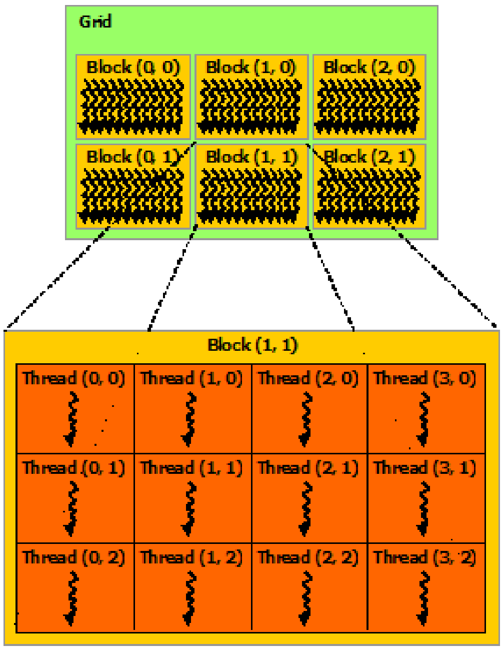


Figure 2.6: *2d grid of blocks*

ones have one copy engine for device to host and one for host to device. This can affect how the calls should be ordered and using the wrong one can make the performance worse than without using asynchronous transfers.
CUBLAS is provides blas functions for cuda. can also be run concurrently with streams
As already mentioned different architectures and even different GPUs differ how they work. Make sure to lookup the different specifics and read the guides for the relevant architecture. NVIDIA uses something called computational capability. It ranges from 1.0, 2.0, 3.0, 3.5 and 5.0. The higher the number the more functions. 1.0 and 2.0 lack some of the things described here.
[32]

2.2.7.1 Efficient CUDA

One of the main criticism against GPUs for general computing purposes is that it is hard to get good performance because it requires good knowledge about details of the GPU architecture, especially the memory architecture. Here are some things that needs to be considered when writing GPU programs.[17, 32, 36]

Maximize parallelism
: Structure the program and the algorithm in such a way that it is as parallel as possible and overlap the serial parts on CPU with calculations on the GPU.[17, 32]

Minimize transfers between host and device
: Moving data between host and device is expensive and should be avoided

if possible. It can be better to run serial parts on the GPU rather than moving the data to the host to do the calculation on the CPU. The bandwidth between host and device is one of the large performance bottlenecks. This can be a problem when the data is to large to fit in the relatively small GPU dram.[32, 36]

Find the optimal number of blocks and threads

There are a lot of things affected by the number of blocks and threads so they should be considered carefully. It is a good idea to parametrize them so that they can be changed for future hardware and varied for optimization. NVIDIA has an occupancy calculator which can be helpful in determining the optimal numbers, however high occupancy does not mean high performance.[32, 36]

The number of blocks should be larger than the number of multiprocessors so that all multiprocessors have at least one block to execute. Having two blocks or more per multiprocessor can be good so that there are blocks that aren't waiting for a __syncthreads() that can be executed. However this is not always possible due to shared memory usage and similar.[36]

The number of threads per block should be a multiplier of 32 but minimum 64. It's also important to remember that multiple concurrent blocks can reside on the same multiprocessor. Too large number of threads in a block and parts of the multiprocessor might be idle since there aren't a block small enough to use those threads. Between 128 and 256 threads is a good place to start.[36]

Streams, concurrent kernels and asynchronous transfers

By using streams it is possible to overlap memory transfers with calculations. This means that the data for the next batch can be transferred while the current batch is calculated and when it is done it can start calculating on the next batch directly after the current one is done. This can hide the time for transfers completely in some situations. However it requires the memory on the host to be pinned. Pinning too much memory can reduce performance of the host.[32]

Operations on a stream is guaranteed to be done in the order they are made however calls on different streams can be executed in any order. Some GPUs allow kernels from different streams to be executed concurrently which can be a big boost if the kernels small enough to fit on the multiprocessors at the same time. Up to 16 or 32 kernels, depending on GPU, can be executed at the same time.

However it requires tweaking of the code that is specific to the GPUs architecture. For instance FERMI architecture only has one queue for the kernels which means that we can get false dependency between kernels. If we have a batch of small kernels and issue them on stream one first and then stream two all the kernels in stream two will be stuck in the queue waiting for the kernels in stream one to finish, this is because the second kernel on stream one has to wait for the first kernel since they are in the same stream. The second kernel then blocks the queue from moving forward so it will not see the kernels on stream two that could be executed. Kepler GPUs on the other hand has several queues so it won't be a problem on those GPUs. It's the same with asynchronous transfers, some GPUs only have one copy engine while others have one engine for device to host and one for host to device.[32, 37, 38]

Use the correct memory on the GPU

Correct use of caches and memory is important for CPU[31]. It's also important on GPUs but it gets more complicated since the caches are smaller and there are several types of memory.

- Register memory is located on the multiprocessor and usually costs zero cycles to access. The multiprocessor splits the available registers over its threads so if there are a lot of threads that uses a lot of variables not all of them will fit in the register. This is why a program sometimes can be faster with lower occupancy.
- Global memory is the main memory of the GPU and is accessible from all threads and blocks. However it is relatively slow to access.
- Shared memory is shared inside a block and is faster than global memory. However it is limited in size.
- Constant memory is small however its cached so it is fast to access. It's read only and best used for small amount of variables that all threads access.
- Local memory is tied to the threads scope, but it is still resides off-chip so it has the same access time as global memory.
- Texture memory is read only and can be faster to access than global memory in some situations. This was more important in older GPUs when global memory wasn't cached.[17, 32]
- Read-only cache is available on Kepler GPUs and uses the same cache as the texture memory. The data as to be read only each multiprocessor can have up to 48kb of space depending on GPU.[38]

Avoid divergence

Each thread in a warp executes the same instruction at the same time so if some of threads diverge the rest will be ideal until they are at the same instruction again. This means it is important to use control structures such as if statements carefully to prevent threads from idling.[32, 36]

Avoid memory bank conflicts when using shared memory

Shared memory is divided into equally-sized memory modules called banks that can be accessed at the same time for higher bandwidth. Bank conflicts occur when separate threads access the same bank. On some GPUs it is fine if all threads access the same bank. Bank conflicts are split into as many conflict-free requests as needed.[32, 36]

Use existing libraries

Instead of writing everything from scratch it is usually a good idea to use already existing libraries. Especially when performance is important and most task are non trivial on GPUs so using an already optimized library is a good idea. Some of the most popular libraries for CUDA are:

- CUBLAS: BLAS implementation for CUDA[39]
- CULAtools: BLAS and LAPACK implementation for CUDA for both dense and sparse matrices[40]
- MAGMA: BLAS and LAPACK implementation among other things that can distribute the work on both CPU and GPU[41]
- Thrust: Template based library that tries to emulate C++ standard library[42]

Avoid slow instructions

There are some instructions that can be slow and should be avoided if possible, for instance type conversion, integer division and modulo. If a function is called with a floating point number that might be used as a double and require a conversion. By putting an f at the end of the number it is told to be single precision float, for instance 0.5f. In some cases it is possible to use bitwise operations instead which is faster.[32, 36]

Restricted pointers can give increased performance

Aliasing can be a problem as mentioned earlier. By using the ___restrict___ keyword on pointers the compiler can be told that no aliasing will occur, however it is up to the programmer to make sure that is the case or there might be unexpected results. Not using aliasing reduces the number of memory accesses the CPU needs to make. However it increases register pressure so it can have an negative effect on performance.[32]

Use fast math functions if precision isn't needed

There are two versions of runtime math functions. The ones of the type funcf() is slower but more accurate than _funcf(). The option -use_fast_math makes the compiler change all the funcf() to _funcf().[36]

2.2.8 Performance Measures

An important part in making fast and efficient programs is to know how fast the program is under certain conditions and which parts of the program are slow. For instance the speed could suddenly drop when we start using to many threads, there might be a bottleneck in the network, and so on.

There are two ways to measure how long a program takes to execute. Execution time, sometimes called wall clock time, is how long real life time the program took. The other is to measure the number of processor cycles spent. A parallel program will have shorter execution time than it is serial version however it will likely have spent more processor cycles due to overhead from communication and initialization of the threads. We are usually interested in execution time however number of cycles can be useful for comparison of algorithms.

speed up is measures how much faster then program is with a certain number of threads compared to the serial version. It's defined as

$$S(p) = \frac{T(1)}{T(p)}$$

Where $T(1)$ is execution time of serial program and $T(p)$ is execution time of parallel program with p threads. Linear speed up is when S(p)=p, it is sometimes called perfect speed up.

Efficiency reflects how efficient the program is using p threads. Linear speed has efficiency 1. It's defined as

$$E(p) = \frac{S(p)}{p} = \frac{T(1)}{pT(p)}$$

Strong scaling refers to how the program handles a fixed problem size and increased number of processors. An program with strong scaling has linear speed up. Weak scaling refers to the execution time of the program when there is a fixed problem size *per processor* and the number of processors is increased.[36]

It can be a good idea to plot these measures while varying p, this can show when bottlenecks occur. Looking at the measures at node level can also be useful to get an idea of how increased number of nodes and therefore increased communication affects the performance.

2.2.8.1 Amdahl's Law and Gustafson's Law

Amdahl's Law is used to find the expected speed up of a system when parts of is parallelised. Simply it says that as the number of processors increases the parts that aren't parallelised will start taking up more and more of the wall clock time and that the speed up for adding more processors will decrease as more and more processors are added and more time is spent relatively on the non parallelised part. It's closely related to strong scaling.[36, 43]

It says that the expected speed up with F fraction of the code parallelised and p threads is

$$S(p) = \frac{1}{(1-F) + \frac{F}{p}(1-F)}$$

As the number of threads grow towards infinity $S(p)$ converges on $\frac{1}{1-F}$. If we have 90% of a code parallelized then even with infinite number of threads we won't get a better speed up than ten.[43]

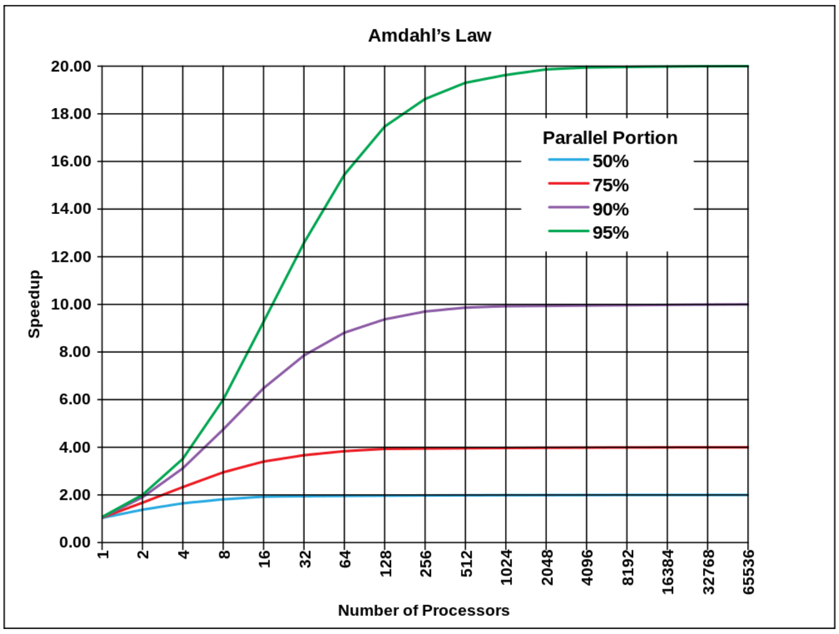


Figure 2.7: *Illustration of Amdahls Law. Wikipedia Commons*

There are limitation to Amdahl's Law since it makes a couple of assumptions.

- The number of executing threads remain constant over the course of the program.
- The parallel portion has perfect speed up. Often not true due to shared resources,eg caches, memory bandwidth, and shared data.
- The parallel portion has infinite scaling, not true due to similar limits as above. More threads will not increase performance after a while or might even decrease it.
- There is no overhead for creation and destruction of threads.
- The length of the serial portion is independent of the number of threads. Often the serial work is to divided the work to the threads, this work will obviously increase as the number of threads go up. More threads can also lead to more communication overhead.
- The serial portion can't be overlapped by the parallel parts. For instance with producer consumer type pattern the consumer could be strictly serial but the time it takes could be overlapped by the parallel producers.

This means it is most accurate with programs that are of the fork-join type, e.g. both serial and parallel parts.

Gustafson's Law is closely related to Amdahl's Law and can in some ways be more accurate than Amdahl's Law. Gustafson's Law makes similar assumptions as Amdahl's Law however it also makes two additional statements. It states that problems tends to expand when provided with more computational power, e.g. increased precision by reducing grid size for simulations, higher frame rate for graphics and so on. The second is that the parallel portion of the program tends to expand faster than the serial part, e.g. for matrix multiplication the initialization scales linearly with the matrix size while the multiplication itself scales as $O(n^3)$. The former means that it is closely related to weak scaling. So in a way it says that the execution time remains constant rather than the amount of data. More precise it says that the expected speed up with p threads and F fraction of the code that is parallel is[44, 36]

$$S(p) = p + (1 - F)(1 - p)$$

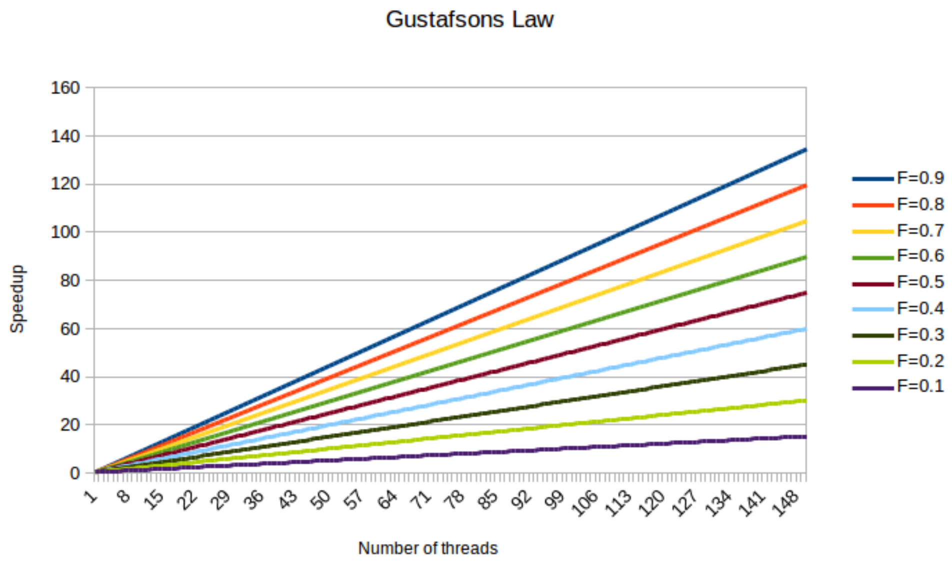


Figure 2.8: *Illustration of Gustafsons Law*

2.2.8.2 Profilers

There are applications called profilers that are made to assess the programs performance and resource consumption. They calculate some of the measures mentioned earlier and they also check hardware usage and how much time the program spends at various parts of the program. This is very useful for finding bottlenecks and other problems in the program. It does not matter if the algorithm is super fast if all the data is stuck in network transfers. The profilers can be hardware dependent so the manufactures usually provided them for their products.[29, 36]

2.3 Software Design

How to design software so it can be maintained over time has been a problem for a long time. Object oriented languages such as C and later Java and C++ arose because of problems with maintaining software.
The messier code will cost productivity in the future when bugs and other problems stack up because of earlier mistakes. All code gather problems overtime, however a well designed system will degenerate a lot slower than one that was designed carelessly. Our code will also be read by ourselves and others later which means readability is important if a future reader is to understand the code.
There are ways to code in such a way that these problems can be mitigated. Most of the things described here goes under the agile development type and the concept of clean code. SOLID is an acronym for five principles for object oriented programming and design. When used together they are intended to make programs that are easy to maintain and extend. As already mentioned readability is important. Correct names will make the code explain itself without other documentation, the code itself is the documentation.

Initial	Principal	Concept
S	Single responsibility principle	A class should only have a single responsibility
O	Open/closed principle	A class should be open for extension but closed for modification
L	Liskov substitution principle	If S is a subtype of T, then objects of type T may be replaced by objects of type S without without altering any of the desirable properties of the program(e.g. work performed)
I	Interface segregation principle	Use several smaller and more specific interfaces instead of one large
D	Dependency inversion principle	Depend on abstractions(e.g. interfaces) not details

Table 2.4: *The five SOLID principles*

Dependency injection is one way to implement dependency inversion principle. Injection is passing the dependency to the dependent object. This is used instead of allowing the dependent object to construct or find the dependency.

2.3.1 Unit Tests and Mocks

Unit testing involves testing the program in small units in isolation. Testing in isolation means that the test should only depend on the part of the program that is tested. If a part of the program is not working properly only its related tests should fail, not other tests for code that depends on it but otherwise works properly. This makes it easier to find errors when they do occur since the tests will pinpoint the unit which does not work.[45]

It is easy to denote test code as less important than the main code but they should be treated as equally important. Tests should also not be an inconvenience to use so they should be easy to run, take reasonable ammount of time to complete and not require any outside interpretation whether they failed or not.[45]

Mocking is to replace a real object with a fake object called a *mock* that for the code is indistinguishable from the real object. This allows one to create situations and test with more control and without depending on the real objects code. The second is important for unit tests since it enables one to test units that normally depend on others in isolation. In the first case it is useful in situations such as when one wants to test a class handling of a rare failure. Such failures can be hard and time consuming to induce. It is then easier to use a mock object that behaves like the failure has occurred. However mocking requires that the code for the class doesn't create the object itself since there is no way to replace the object with the mock. This is one of the reasons why dependency injection should be use.[45]

2.3.2 Design Patterns

A *design pattern* in software design is a reusable general solution to a common recurring problem in a given context. It is templates and structures to solve the problem, however it is not code. However they are partially dependent on the programming language since different languages have different features and limitations.

The *? pattern*
Concurrency patterns are design patterns in contexts which involves concurrency.
Consumer producer is a pattern for when there are a number of consumers/workers and producers. They share a common queue for products. The producer generates some product and put it into the queue, while the the consumers consume the products. The problem is that the producers should not add to an non empty slot and that each product should only be consumed once.

Chapter 3

Algorithm

3.1 Current algorithm

JEIRA uses Java and Javas built-in functions for concurrency.
Producer consumer pattern

3.2 CuEira

Producer consumer pattern ?
Strucuture picture here.
unrolling+padding compilers

Chapter 4

Results

Chapter 5

Discussion and Conclusions

Chapter 6

Outlook

cublas not that much harder than regular blas. A wrapper could enable simple and easy use without losing too much features. own kernels more than simple elemntwise, probably hard to optimize. unknowish

Chapter 7

Appendix

7.1 A List of Variables

OR Odds ratio

RR_{ij} Relative risk with exposure i j?

X Predictors

Y The outcome, binary variable

$\pi(x)$ P(Y=1)

7.2 B List of Abbrivations

MAF
Minor allele frequency, the frequency of the least common allele.

RF
Random Forest

Bibliography

[1] H. J. Cordell, "Detecting gene–gene interactions that underlie human diseases," *Nature Reviews Genetics*, vol. 10, no. 6, pp. 392–404, 2009.

[2] S. J. Winham and J. M. Biernacka, "Gene–environment interactions in genome-wide association studies: current approaches and new directions," *Journal of Child Psychology and Psychiatry*, vol. 54, no. 10, pp. 1120–1134, 2013.

[3] B. Ding, H. Källberg, L. Klareskog, L. Padyukov, and L. Alfredsson, "Geira: gene-environment and gene-gene interaction research application," *European Journal of Epidemiology*, vol. 26, no. 7, pp. 557–561, 2011.

[4] H. Mahdi, B. A. Fisher, H. Källberg, D. Plant, V. Malmström, J. Rönnelid, P. Charles, B. Ding, L. Alfredsson, L. Padyukov, *et al.*, "Specific interaction between genotype, smoking and autoimmunity to citrullinated α-enolase in the etiology of rheumatoid arthritis," *Nature genetics*, vol. 41, no. 12, pp. 1319–1324, 2009.

[5] K. Rothman and S. Greenland, *Modern Epidemiology.* London: Lippincott Williams and Wilkins, 1998.

[6] C. Mann, "Observational research methods. research design ii: cohort, cross sectional, and case-control studies," *Emergency Medicine Journal*, vol. 20, no. 1, pp. 54–60, 2003.

[7] M. Fareed and M. Afzal, "Single nucleotide polymorphism in genome-wide association of human population: A tool for broad spectrum service," *Egyptian Journal of Medical Human Genetics*, vol. 14, no. 2, pp. 123 – 134, 2013.

[8] B. Goudey, D. Rawlinson, Q. Wang, F. Shi, H. Ferra, R. M. Campbell, L. Stern, M. T. Inouye, C. S. Ong, and A. Kowalczyk, "Gwis-model-free, fast and exhaustive search for epistatic interactions in case-control gwas," *BMC genomics*, vol. 14, no. Suppl 3, p. S10, 2013.

[9] G. Fang, M. Haznadar, W. Wang, H. Yu, M. Steinbach, T. R. Church, W. S. Oetting, B. Van Ness, and V. Kumar, "High-order snp combinations associated with complex diseases: efficient discovery, statistical power and functional interactions," *PloS one*, vol. 7, no. 4, p. e33531, 2012.

[10] S. Leem, H.-h. Jeong, J. Lee, K. Wee, and K.-A. Sohn, "Fast detection of high-order epistatic interactions in genome-wide association studies using information theoretic measure," *Computational Biology and Chemistry*, 2014.

[11] K. J. Rothman, *Epidemiology: an introduction.* Oxford University Press, 2002.

[12] A. Gyenesei, J. Moody, A. Laiho, C. A. Semple, C. S. Haley, and W.-H. Wei, "Biforce toolbox: powerful high-throughput computational analysis of gene–gene interactions in genome-wide association studies," *Nucleic acids research*, vol. 40, no. W1, pp. W628–W632, 2012.

[13] L. S. Yung, C. Yang, X. Wan, and W. Yu, "Gboost: a gpu-based tool for detecting gene–gene interactions in genome–wide case control studies," *Bioinformatics*, vol. 27, no. 9, pp. 1309–1310, 2011.

[14] Z. Zhu, X. Tong, Z. Zhu, M. Liang, W. Cui, K. Su, M. D. Li, and J. Zhu, "Development of gmdr-gpu for gene-gene interaction analysis and its application to wtccc gwas data for type 2 diabetes," *PloS one*, vol. 8, no. 4, p. e61943, 2013.

[15] S. Lee, M.-S. Kwon, I.-S. Huh, and T. Park, "Cuda-lr: Cuda-accelerated logistic regression analysis tool for gene-gene interaction for genome-wide association study," in *Bioinformatics and Biomedicine Workshops (BIBMW), 2011 IEEE International Conference on*, pp. 691–695, IEEE, 2011.

[16] S. Chikkagoudar, K. Wang, and M. Li, "Genie: a software package for gene-gene interaction analysis in genetic association studies using multiple gpu or cpu cores," *BMC research notes*, vol. 4, no. 1, p. 158, 2011.

[17] J. Poznanovic, "Plinkgpu: A framework for gpu acceleration of whole genome data analysis," Master's thesis, School of Informatics, University of Edinburgh, 2010.

[18] M. D. Ritchie, L. W. Hahn, N. Roodi, L. R. Bailey, W. D. Dupont, F. F. Parl, and J. H. Moore, "Multifactor-dimensionality reduction reveals high-order interactions among estrogen-metabolism genes in sporadic breast cancer," *The American Journal of Human Genetics*, vol. 69, no. 1, pp. 138–147, 2001.

[19] X. Wan, C. Yang, Q. Yang, H. Xue, X. Fan, N. L. Tang, and W. Yu, "Boost: A fast approach to detecting gene-gene interactions in genome-wide case-control studies," *The American Journal of Human Genetics*, vol. 87, no. 3, pp. 325–340, 2010.

[20] Y. Zhang and J. S. Liu, "Bayesian inference of epistatic interactions in case-control studies," *Nature genetics*, vol. 39, no. 9, pp. 1167–1173, 2007.

[21] J. M. Bland and D. G. Altman, "Multiple significance tests: the bonferroni method," *Bmj*, vol. 310, no. 6973, p. 170, 1995.

[22] A. Agresti, *Categorical data analysis.* John Wiley & Sons, Second ed., 2002.

[23] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[24] D. F. Schwarz, I. R. König, and A. Ziegler, "On safari to random jungle: a fast implementation of random forests for high-dimensional data," *Bioinformatics*, vol. 26, no. 14, pp. 1752–1758, 2010.

[25] S. J. Winham, C. L. Colby, R. R. Freimuth, X. Wang, M. de Andrade, M. Huebner, and J. M. Biernacka, “Snp interaction detection with random forests in high-dimensional genetic data,” *BMC bioinformatics*, vol. 13, no. 1, p. 164, 2012.

[26] J. Von Neumann, “First draft of a report on the edvac,” *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.

[27] W. F. Gilreath and P. A. Laplante, *Computer Architecture: A Minimalist Perspective: Dynamics and Sustainability.* Springer, 2003.

[28] S. Almeida, “An introduction to high performance computing,” *International Journal of Modern Physics A*, vol. 28, no. 22n23, p. 1340021, 2013.

[29] G. Hager and G. Wellein, *Introduction to high performance computing for scientists and engineers.* CRC Press, 2010.

[30] “TOP500 Supercomputer Site.” `http://www.top500.org`.

[31] U. Drepper, “What every programmer should know about memory,” 2007.

[32] NVIDIA, *NVIDIA CUDA C Programming Guide*, v5.5 ed., July 2013.

[33] NVIDIA, “Nvidia tesla-kepler product description.” `http://www.nvidia.com/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf`.

[34] R. Jiang, F. Zeng, W. Zhang, X. Wu, and Z. Yu, “Accelerating genome-wide association studies using cuda compatible graphics processing units,” pp. 70–76, 2009.

[35] D. B. Kirk and W. H. Wen-mei, *Programming massively parallel processors: a hands-on approach.* Newnes, 2012.

[36] NVIDIA, *NVIDIA CUDA C Best Practices Guide*, v5.5 ed., July 2013.

[37] Mark Harris, “How to Overlap Data Transfers in CUDA C/C++.” `http://devblogs.nvidia.com/parallelforall/how-overlap-data-transfers-cuda-cc/`.

[38] NVIDIA, “Kepler Tuning Guide.” `http://docs.nvidia.com/cuda/kepler-tuning-guide`.

[39] NVIDIA, *CUBLAS Library User Guide*, v5.5 ed., July 2013.

[40] “CULA Tools: GPU Accelerated Linear Algebra,” 2010.

[41] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, “Dense linear algebra solvers for multicore with gpu accelerators,” in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pp. 1–8, IEEE, 2010.

[42] J. Hoberock and N. Bell, “Thrust: A parallel template library,” 2010. Version 1.7.0.

[43] X.-H. Sun and Y. Chen, “Reevaluating amdahlâĂŹs law in the multicore era,” *Journal of Parallel and Distributed Computing*, vol. 70, no. 2, pp. 183–188, 2010.

[44] J. L. Gustafson, “Reevaluating amdahl’s law,” *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[45] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2008.